Algorithmic Algorithm Invention in the Theorema Project

Bruno Buchberger

Talk at AIT (Algorithmic Information Theory)
Vaasa, Finland, May 16-18, 2005

RISC (Research Institute for Symbolic Computation)
Johannes Kepler University

RICAM (Radon Institute for Computational and Applied Mathematics)
Austrian Academy of Science

Linz, Austria

The Theorema Project

An "Algorithm" for Algorithm Synthesis

Synthesis of a Gröbner Bases Algorithm

Conclusion

The Theorema Project

An "Algorithm" for Algorithm Synthesis

Synthesis of a Gröbner Bases Algorithm

Conclusion

The Objective of Theorema

A system that supports (partially automates) the entire "mathematical theory exploration" process:

Starting from some given mathematical concepts and mathematical knowledge on these concepts within a uniform logical language (predicate logic),

- invent definitions (axioms) of new concepts
- invent and prove / disprove propositions on these concepts
- invent problems
- invent and verify algorithms for problems
- store the definitions, propositions, problems, algorithms in structured knowledge libraries

The *Theorema* group: B. B. (leader), T. Jebelean, T. Kutsia, F. Piroi, M. Rosenkranz, W. Windsteiger, and PhD students.

Emphasis of this Talk

(Partially) automated invention of algorithms from problem specifications.

Main idea: invention from failing proofs.

A Simple Example of the Theorema Proof Style

```
Definition["addition", any[m, n],
    m + 0 = m          " +0:"
    m + n⁺ = (m + n)⁺  " + .:"]
```

```
Proposition["left zero", any[m, n],
    0 + n = n  "0+"]
```

```
Prove[Proposition["left zero"],
    using → ⟨Definition["addition"]⟩,
    by → NNEqIndProver,

    ProverOptions → {TermOrder → LeftToRight},
 transformBy → ProofSimplifier, TransformerOptions → {branches → {Proved}}];
```


Example: Failing Proof

```
Proposition["commutativity of addition", any[m, n],
    m + n = n + m  " + = "]
```

```
Prove[Proposition["commutativity of addition"],
    using → ⟨Definition["addition"]⟩,
    by → NNEqIndProver,

    ProverOptions → {TermOrder → LeftToRight}, transformBy → ProofSimplifier,
 TransformerOptions → {branches → {Proved, Failed}}];
```


Example: (Automatic) Inventions from Failing Proofs

Theorema tool "Cascade":

```
Prove[Proposition["commutativity of addition"],
    using → Definition["addition"],
  by → Cascade[NNEqIndProver, ConjectureGenerator],
  ProverOptions → {TermOrder → LeftToRight}];
```

This idea (in a slight generalization) is also an essential ingredient for algorithm synthesis in *Theorema*.

The Theorema Project

An "Algorithm" for Algorithm Synthesis

Synthesis of a Gröbner Bases Algorithm

Conclusion

The Algorithm Invention ("Synthesis") Problem

Given a problem specification P (in predicate logic), find an algorithm A such that

$$\forall_x \mathbf{P}[x, \mathbf{A}[x]].$$

Examples of specifications P:

```
P[x, y] ⇔ is-greater[x, y]
P[x, y] ⇔ is-sorted-version[x, y]
P[x, y] ⇔ has-derivative[x, y]
P[x, y] ⇔ are-factors-of[x, y]
P[x, y] ⇔ is-Gröbner-basis[x, y]
....
```

A general algorithm S for "all" P cannot exist but ...

There is a rich literature on algorithm synthesis methods.

The "Lazy Thinking" Method for Algorithm Synthesis (BB 2001): Sketch

Given a problem specification P

- consider various "algorithm schemes" for A, e.g.
  $\mathbf{A}[\langle\rangle] = \mathbf{c}$
  $\forall_x \mathbf{A}[\langle x\rangle] = \mathbf{s}[\langle x\rangle]$
  $\forall_{x,y} (\mathbf{A}[\langle x, y\rangle] = \mathbf{i}[x, \mathbf{A}[\langle y\rangle]])$
- and try to prove (automatically) $\forall_x \mathbf{P}[x, \mathbf{A}[x]].$
- This proof will normally fail because nothing is known on the unspecified sub-algorithms in the algorithm scheme.
- From the temporary assumptions and goals in the failing proof situation (automatically) generate such specifications for the unspecified sub-algorithms that would make the proof possible.

Now, apply the method recursively to the auxiliary functions.

Example: Synthesis of Merge-Sort [BB et al. 2003]

Problem: Synthesize "sorted" such that

$$\forall_{x} \; \text{is-sorted-version}[x, \text{sorted}[x]].$$

("Correctness Theorem")

Knowledge on Problem:

$$\forall_{x,y} \left(\text{is-sorted-version}[x, y] \Leftrightarrow \begin{array}{l} \text{is-sorted}[y] \\ \text{is-permuted-version}[x, y] \end{array} \right)$$

$$\text{is-sorted}[\langle\rangle]$$

$$\forall_{x} \; \text{is-sorted}[\langle x \rangle]$$

$$\forall_{x,y,z} \left(\text{is-sorted}[\langle x, y, z \rangle] \Leftrightarrow \begin{array}{l} x \geq y \\ \text{is-sorted}[\langle y, z \rangle] \end{array} \right)$$

etc.

An Algorithm Scheme: Divide and Conquer

$$\forall_{x} \left(\text{sorted}[x] = \begin{cases} \text{special}[x] & \Leftarrow \; \text{is-trivial-tuple}[x] \\ \text{merge}[\text{sorted}[\text{left-split}[x]], \text{sorted}[\text{right-split}[x]]] & \Leftarrow \; \text{otherwise} \end{cases} \right)$$

We Now Start Proving the Correctness Theorem and Analyze the Failing Proof: see notebooks with failing proofs.

Automated Invention of Sufficient Specifications for the Subalgorithms

A simple (but amazingly powerful) rule:

Collect temporary assumptions T[x0, ... A [ ], ...]

and temporary goals G[x0, ...m [A [ ]]]

and produces specification

$$\forall_{x,\ldots}\ \forall_{Y,\ldots}\ \left(T[x,\ldots Y,\ldots] \Longrightarrow G[Y,\ldots m\ [Y]]\right).$$

Details: see papers [BB 2003] and example.

The Result of Applying Lazy Thinking in the Sorting Example

Lazy Thinking, automatically (in approx. 2 minutes on a laptop using the *Theorema* system), finds the following specifications for the sub-algorithms that provenly guarantee the correctness of the above algorithm (scheme):

$$\forall_{x}\ (\text{is-trivial-tuple}[x] \Rightarrow \text{special}[x] = x)$$

$$\forall_{y,z}\ \left(\begin{array}{l}\text{is-sorted}[y]\\ \text{is-sorted}[z]\end{array} \Rightarrow \begin{array}{l}\text{is-sorted}[\text{merged}[y, z]]\\ \text{merged}[y, z] \approx (y \asymp z)\end{array}\right)$$

$$\forall_{x}\ (\text{left–split}[x] \asymp \text{right–split}[x] \approx x)$$

Note: the specifications generated are not only sufficient but natural !

What Do We Have Now?

- **Case A:** We find algorithms special, merged, left–split, right–split in our knowledge base for which the properties specified above are already contained in the knowledge base or can be derived from the knowledge base.

  In this case, we are done, i.e. we have synthesized a sorting algorithm.

- **Case B:** We do not find algorithms special, merged, left–split, right–split in our knowledge base for which the properties specified can be proved.

  In this case, we apply Lazy Thinking again in order to synthesize appropriate special, merged, left–split, right–split

  until we arrive at sub-sub-...-algorithms in our knowledge base (e.g. the basic operations of tuple theory like append, prepend etc.)

Case B can be avoided, if we proceed systematically bottom-up ("complete theory exploration" in layers).

Example: Synthesis of Insertion-Sort

Synthesize A such that

$$\forall_{x} \; \text{is-sorted-version}[x, A[x]].$$

Algorithm Scheme: "simple recursion"

$$A[\langle\rangle] = c$$
$$\forall_{x} \; A[\langle x\rangle] = s[\langle x\rangle]$$
$$\forall_{x,y} \; (A[\langle x, y\rangle] = i[x, A[\langle y\rangle]])$$

Lazy Thinking, automatically (in approx. 2 minutes on a laptop using the *Theorema* system), finds the following specifications for the auxiliary functions

$$c = \langle\rangle$$
$$\forall_{x} \; (s[\langle x\rangle] = \langle x\rangle)$$
$$\forall_{x,y} \left(\text{is-sorted}[\langle y\rangle] \Rightarrow \begin{array}{l} \text{is-sorted}[i[x, \langle y\rangle]] \\ i[\langle x, y\rangle] \approx (x \frown \langle y\rangle) \end{array} \right)$$

How Far Can We Go With the Method ?

Can we automatically synthesize algorithms for non-trivial problems?

Example of a non-trivial problem: construction of Gröbner bases. What is "non-trivial"?

Main algorithmic idea of Gröbner bases theory: The "S-polynomials" together with the S-polynomial theorem.

Hence, question: Can Lazy Thinking automatically invent the notion of S-polynomial and automatically deliver the S-polynomial theorem.

The Theorema Project

An "Algorithm" for Algorithm Synthesis

Synthesis of a Gröbner Bases Algorithm

Conclusion

The Problem of Constructing Gröbner Bases

Find algorithm Gb such that

$$\forall_{\text{is-finite}[F]} \begin{pmatrix} \text{is-finite}[\,\text{Gb}[F]\,] \\ \text{is-Gröbner-basis}[\,\text{Gb}[F]] \\ \text{ideal}[F] = \text{ideal}[\,\text{Gb}[F]]\,. \end{pmatrix}$$

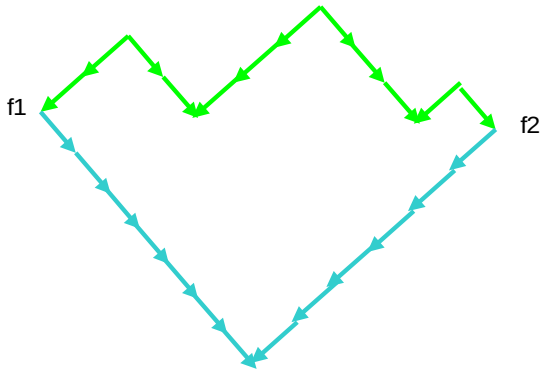
Definitions [BB 1965, 1970] (a definition, which is equivalent to the one I gave yesterday):

$$\text{is-Gröbner-basis}[G] \Leftrightarrow \text{is-confluent}[\,\to_G\,].$$

$\to_G$ ... a division step (of yesterday).

Confluence of Division $\to_G$

$$\text{is-confluent}[\,\to\,] : \Leftrightarrow \forall_{f1,f2} (f1 \leftrightarrow^* f2 \Rightarrow f1 \downarrow^* f2)$$



Knowledge on the Concepts Involved

$$h1 \to_G h2 \;\Rightarrow\; p \,.\, h1 \to_G p \,.\, h2$$

etc.

Algorithm Scheme "Critical Pair / Completion"

$$A[F] = A[F, pairs[F]]$$
$$A[F, \langle\rangle] = F$$

$$A[F, \langle\langle g1, g2\rangle, p\rangle] = \text{where}\left[f = lc[g1, g2],\ h1 = trd[rd[f, g1], F],\ h2 = trd[rd[f, g2], F],\ \begin{cases} A[F, \langle p\rangle] & \Leftarrow\ h1 = h2 \\ A\left[F \frown df[h1, h2], \langle p\rangle \asymp \left\langle \langle F_k, df[h1, h2]\rangle \;\middle|\; {\scriptstyle k=1,\ldots,|F|} \right\rangle\right] & \Leftarrow\ \text{otherwise} \end{cases}\right]$$

This scheme can be tried in any domain, in which we have a reduction operation rd that depends on sets F of objects and a Noetherian relation $\succ$ which interacts with rd in the following natural way:

$$\underset{f,g}{\forall}\ (f \succeq rd[f, g]).$$

The Essential Problem

The problem of synthesizing a Gröbner bases algorithm can now be also stated by asking whether starting with the proof of

$$\underset{F}{\forall}\ \begin{pmatrix} \text{is-finite}[\,A[F]\,] \\ \text{is-Gröbner-basis}[\,A[F]] \\ ideal[F] = ideal[\,A[F]]. \end{pmatrix}$$

we can *automatically* produce the idea that

```
lc[g1, g2] = lcm[lp[g1], lp[g2]]
```

and

```
df[h1, h2] = h1 - h2
```

and prove that the idea is correct.

Now Start the (Automated) Correctness Proof

With current theorem proving technology, in the *Theorema* system, the proof attempt could be done automatically. (Not yet fully implemented.)

Details

It should be clear that, if the algorithm terminates, the final result is a finite set (of polynomials) G that has the property

$$\forall_{g1,g2 \in G} \left(\text{where}\left[f = lc[g1, g2],\ h1 = trd[rd[f, g1], F],\ h2 = trd[rd[f, g2], F],\ \bigvee \left\{ \begin{matrix} h1 = h2 \\ df[h1, h2] \in G \end{matrix} \right. \right] \right).$$

We now try to prove that, if G has this property, then

```
is-finite[G],
ideal[F] = ideal[G],
is-Gröbner-basis[G],
  i.e. is-Church-Rosser[ →G ].
```

Here, we only deal with the third, most important, property.

Using Available Knowledge

Using Newman's lemma and some elementary properties it can be shown that it is sufficient to prove

$$\text{is-Church-Rosser}[\ \to_G\ ] \Leftrightarrow \forall_p \forall_{f1,f2} \left(\left\{ \begin{matrix} p \to f1 \\ p \to f2 \end{matrix} \right. \right) \Rightarrow f1 \downarrow^* f2 \Big).$$

The (Automated) Proof Attempt

Let now the power product p and the polynomials f1, f2 be arbitary but fixed and assume

$$\left\{ \begin{matrix} p \to_G f1 \\ p \to_G f2. \end{matrix} \right.$$

We have to find a polyonomial g such that

$$\begin{aligned} &f1 \to_G^* g, \\ &f2 \to_G^* g. \end{aligned}$$

From the assumption we know that there exist polynomials g1 and g2 in G such that

$$\begin{aligned} &lp[g1] \mid p, \\ &f1 = rd[p, g1], \\ &lp[g2] \mid p, \\ &f2 = rd[p, g2]. \end{aligned}$$

From the final situation in the algorithm scheme we know that for these g1 and g2

$$\bigvee \left\{ \begin{matrix} h1 = h2 \\ df[h1, h2] \in G, \end{matrix} \right.$$

where

$$\begin{aligned} &h1 := trd[f1', G], \ f1' := rd[lc[g1, g2], g1], \\ &h2 := trd[f2', G], \ f2' := rd[lc[g1, g2], g2]. \end{aligned}$$

Case h1=h2

$$\mathtt{lc[g1,\ g2]} \to_{\mathtt{g1}} \mathtt{rd[lc[g1,\ g2],\ g1]} \to_{\mathtt{G}}{}^{*} \mathtt{trd[rd[lc[g1,\ g2],\ g1],\ G]} = \mathtt{trd[rd[lc[g1,\ g2],\ g2],\ G]} \leftarrow_{\mathtt{G}}{}^{*} \mathtt{rd[lc[g1,\ g2],\ g2]} \leftarrow_{\mathtt{g2}} \mathtt{lc[g1,\ g2]}.$$

(Note that here we used the requirements rd[lc[g1,g2],g1]≺lc[g1,g2] and rd[lc[g1,g2],g2]≺lc[g1,g2].)

Hence, by elementary properties of polynomial reduction,

$$\forall_{\mathtt{a,q}} \left(\mathtt{a\ q\ lc[g1,\ g2]} \to_{\mathtt{g1}} \mathtt{a\ q\ rd[lc[g1,\ g2],\ g1]} \to_{\mathtt{G}}{}^{*} \mathtt{a\ q\ trd[rd[lc[g1,\ g2],\ g1],\ G]} = \mathtt{a\ q\ trd[rd[lc[g1,\ g2],\ g2],\ G]} \leftarrow_{\mathtt{G}}{}^{*} \mathtt{a\ q\ rd[lc[g1,\ g2],\ g2]} \leftarrow_{\mathtt{g2}} \mathtt{a\ q\ lc[g1,\ g2]}\right).$$

Now we are stuck in the proof.

Now Use the Specification Generation Algorithm

Using the above specification generation rule, we see that we could proceed successfully with the proof if lc[g1,g2] satisfied the following requirement

$$\forall_{\mathtt{p,g1,g2}} \left(\left(\left\{\begin{matrix}\mathtt{lp[g1]} \mid \mathtt{p} \\ \mathtt{lp[g2]} \mid \mathtt{p}\end{matrix}\right\}\right) \Rightarrow \left(\exists_{\mathtt{a,q}} \left(\mathtt{p} = \mathtt{a\ q\ lc[g1,\ g2]}\right)\right)\right), \quad \text{(lc requirement)}$$

With such an lc, we then would have

$$\mathtt{p} \to_{\mathtt{g1}} \mathtt{rd[p,\ g1]} = \mathtt{a\ q\ rd[lc[g1,\ g2],\ g1]} \to_{\mathtt{G}}{}^{*} \mathtt{a\ q\ trd[rd[lc[g1,\ g2],\ g1],\ G]} = \mathtt{a\ q\ trd[rd[lc[g1,\ g2],\ g2],\ G]} \leftarrow_{\mathtt{G}}{}^{*} \mathtt{a\ q\ rd[lc[g1,\ g2],\ g2]} = \mathtt{rd[p,\ g2]} \leftarrow_{\mathtt{g2}} \mathtt{p}$$

and, hence,

$$\mathtt{f1} \to_{\mathtt{G}}{}^{*} \mathtt{a\ q\ trd[rd[lc[g1,\ g2],\ g1],\ G]},$$

$$\mathtt{f2} \to_{\mathtt{G}}{}^{*} \mathtt{a\ q\ trd[rd[lc[g1,\ g2],\ g1],\ G]},$$

i.e. we would have found a suitable g.

Summarize the (Automatically Generated) Specifications of the Subalgorithm lc

(lc requirement), which also could be written in the form:

$$\forall_{p,g1,g2} \left(\left\{ \begin{matrix} \text{lp}[g1] \mid p \\ \text{lp}[g2] \mid p \end{matrix} \right\} \Rightarrow (\text{lc}[g1, g2] \mid p) \right),$$

and the requirements:

```
rd[lc[g1, g2], g1] ≺ lc[g1, g2],
rd[lc[g1, g2], g2] ≺ lc[g1, g2],
```

which, in the case of the domain of polynomials, are equivalent to

```
lp[g1] | lc[g1, g2],
lp[g2] | lc[g1, g2].
```


A Suitable lc

```
lcp[g1, g2] = lcm[lp[g1], lp[g2]]
```

is a suitable function that satisfies the above requirements.

Heureka! The crucial function lc (the "critical pair" function) in the critical pair / completion algorithm scheme has been "automatically" synthesized!

Case h1≠h2

In this case, df[h1,h2]∈G:

In this part of the proof we are basically stuck right at the beginning.

We can try to reduce this case to the first case, which would generate the following requirement

$$\forall_{h1,h2}\ (h1 \downarrow_{\{df[h1,h2]\}} * h2)\ \ (df\ requirement).$$

Looking to the Knowledge Base for a Suitable df

(Looking to the knowledge base of elementary properties of polynomial reduction, it is now easy to find a function df that satifies (df requirement), namely

$$df[h1, h2] = h1 - h2,$$

because, in fact,

$$\forall_{f,g}\ (f \downarrow_{\{f-g\}} * g).$$

Heureka! The function df (the "completion" function) in the critical pair / completion algorithm scheme has been "automatically" synthesized!)

Namely,

35 of 42

The Theorema Project

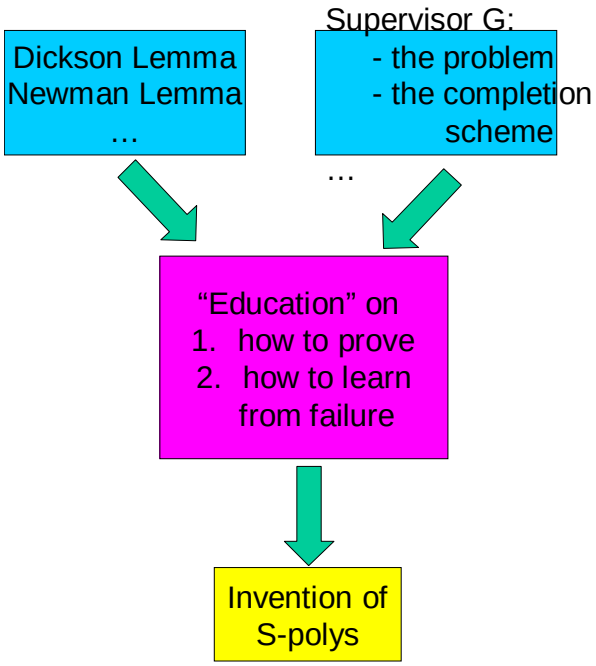
An "Algorithm" for Algorithm Synthesis

Synthesis of a Gröbner Bases Algorithm

Conclusion

36 of 42

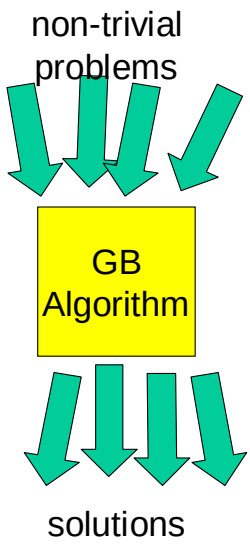
A way of looking at it ("what would have happened if ..."):



Pairs idea?

$$A[F, \langle\langle g1, g2\rangle, p\rangle] = \text{where}\left[f = lc[g1, g2],\ h1 = trd[rd[f, g1], F],\ h2 = trd[rd[f, g2], F],\ \begin{cases} A[F, \langle p\rangle] & \Leftarrow\ h1 = h2 \\ A\left[F \frown df[h1, h2], \langle p\rangle \asymp \left\langle \langle F_k, df[h1, h2]\rangle \Big|_{k=1,\ldots,|F|} \right\rangle\right] & \Leftarrow\ \text{otherwise} \end{cases}\right]$$

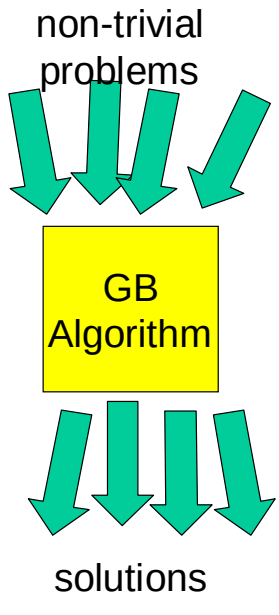
Bad and Good for Me

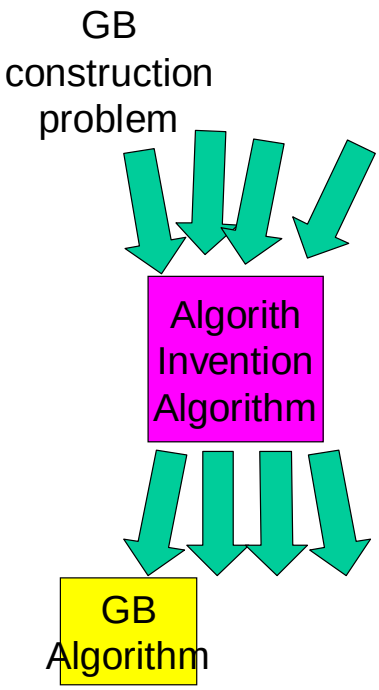


Good for me !

38 of 42

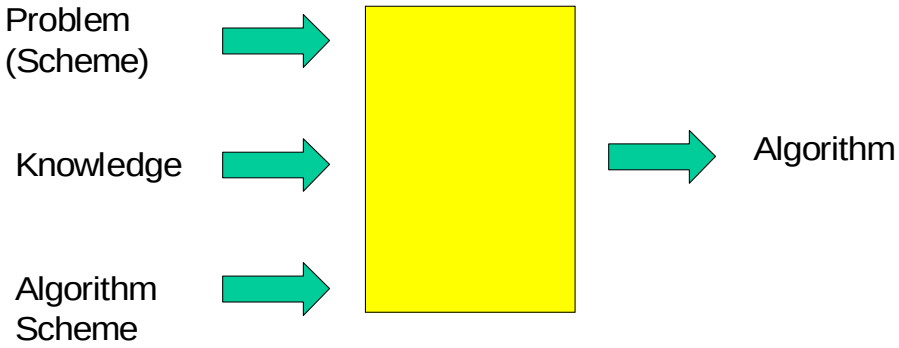
Bad and Good for Me





Good for me ! Good and bad for me !

Research Topics



- **Libraries** of algorithm **schemes**.

  More generally, libraries of definition, theorem, problem, and algorithm schemes.

- **Case studies** of problem (schemes), knowledge, algorithm schemes and how they produce algorithms.

- Improved **algorithms for generating problem specifications** from failing proofs.

Special Semester on Gröbner Bases at RICAM and RISC

Softwarepark Hagenberg

42 of 42

References

■ On Gröbner Bases

[Buchberger 1970]

B. Buchberger. Ein algorithmisches Kriterium für die Lösbarkeit eines algebraischen Gleichungssystems (An Algorithmical Criterion for the Solvability of Algebraic Systems of Equations). Aequationes mathematicae 4/3, 1970, pp. 374-383. (English translation in: [Buchberger, Winkler 1998], pp. 535 -545.) Published version of the PhD Thesis of B. Buchberger, University of Innsbruck, Austria, 1965.

[Buchberger 1998]

B. Buchberger. Introduction to Gröbner Bases. In: [Buchberger, Winkler 1998], pp.3-31.

[Buchberger, Winkler, 1998]

B. Buchberger, F. Winkler (eds.). Gröbner Bases and Applications, Proceedings of the International Conference "33 Years of Gröbner Bases", 1998, RISC, Austria, London Mathematical Society Lecture Note Series, Vol. 251, Cambridge University Press, 1998.

[Becker, Weispfenning 1993]

T. Becker, V. Weispfenning. Gröbner Bases: A Computational Approach to Commutative Algebra, Springer, New York, 1993.

■ On Mathematical Knowledge Management

B. Buchberger, G. Gonnet, M. Hazewinkel (eds.)

Mathematical Knowledge Management.

Special Issue of Annals of Mathematics and Artificial Intelligence, Vol. 38, No. 1-3, May 2003, Kluwer Academic Publisher, 232 pages.

A.Asperti, B. Buchberger, J.H.Davenport (eds.)

Mathematical Knowledge Management.

Proceedings of the Second International Conference on Mathematical Knowledge Management (MKM 2003), Bertinoro, Italy, Feb.16-18, 2003, Lecture Notes in Computer Science, Vol. 2594, Springer, Berlin-Heidelberg-NewYork, 2003, 223 pages.

A.Asperti, G.Bancerek, A.Trybulec (eds.).

Proceedings of the Third International Conference on Mathematical Knowledge Management, MKM 2004,

Bialowieza, Poland, September 19-21, 2004, Lecture Notes in Computer Science, Vol. 3119, Springer, Berlin-Heidelberg-NewYork, 2004

■ On Theorema

[Buchberger et al. 2000]

B. Buchberger, C. Dupre, T. Jebelean, F. Kriftner, K. Nakagawa, D. Vasaru, W. Windsteiger. The Theorema Project: A Progress Report. In: M. Kerber and M. Kohlhase (eds.), Symbolic Computation and Automated Reasoning (Proceedings of CALCULEMUS 2000, Symposium on the Integration of Symbolic Computation and Mechanized Reasoning, August 6-7, 2000, St. Andrews, Scotland), A.K. Peters, Natick, Massachusetts, ISBN 1-56881-145-4, pp. 98-113.

■ On Theory Exploration and Algorithm Synthesis

[Buchberger 2000]

B. Buchberger. Theory Exploration with *Theorema*.

Analele Universitatii Din Timisoara, Ser. Matematica-Informatica, Vol. XXXVIII, Fasc.2, 2000, (Proceedings of SYNASC 2000, 2nd International Workshop on Symbolic and Numeric Algorithms in Scientific Computing, Oct. 4-6, 2000, Timisoara, Rumania, T. Jebelean, V. Negru, A. Popovici eds.), ISSN 1124-970X, pp. 9-32.

[Buchberger 2003]

B. Buchberger. Algorithm Invention and Verification by Lazy Thinking.

In: D. Petcu, V. Negru, D. Zaharie, T. Jebelean (eds), Proceedings of SYNASC 2003 (Symbolic and

Numeric Algorithms for Scientific Computing, Timisoara, Romania, October 1–4, 2003), Mirton Publishing, ISBN 973–661–104–3, pp. 2–26.

[Buchberger, Craciun 2003]

B. Buchberger, A. Craciun. Algorithm Synthesis by Lazy Thinking: Examples and Implementation in Theorema. in: Fairouz Kamareddine (ed.), Proc. of the Mathematical Knowledge Management Workshop, Edinburgh, Nov. 25, 2003, Electronic Notes on Theoretical Computer Science, volume dedicated to the MKM 03 Symposium, Elsevier, ISBN 044451290X, to appear.

[Buchberger 2004]

B. Buchberger.

Towards the Automated Synthesis of a Gröbner Bases Algorithm.

RACSAM (Review of the Royal Spanish Academy of Science), Vol. 98/1, to appear, 10 pages.